Implementing Real-Time Data Analytics to RTI Connext Systems with JupyterLab

Introduction

The importance of data analytics in today's world is undeniable. It provides important insight into the connections between various aspects of our data, broadens our knowledge and understanding, helps us form more sophisticated decisions and uncovers crucial correlations that might carry very important information regarding our systems and processes.

Implementing data analytics to RTI Connext [1] applications can be highly beneficial to most projects. However, choosing the right approach can be a daunting task, as new technologies spring up every day in this ever-changing field.

There are many viable choices and approaches – both free and paid-for solutions – that can be considered, such as the Telegraf plugin for Connext DDS [2]. In this article, we will be looking at the utilization of one of the most popular data analytics environments, JupyterLab [3].

The JupyterLab environment

JupyterLab is a free and open-source software, based on open standards providing interactive computing across various programming languages. It gives a very powerful environment for data science, scientific computing, and machine learning where code, documentation and visualization can reside in the same document allowing developers to focus on analytics without any distractions. Moreover, thanks to its modular design, it is highly customizable and can be tailored to any use-case with the help of ready-made and custom extensions.

The most popular programming language in JupyterLab is Python [4], – as it is the most popular language in the field of data analytics and scientific computing – however, it supports over 40 languages with the help of different customizable kernels. Python, with its rich ecosystem and millions of additional packages including Pandas [5], NumPy [6], Scikit [7], TensorFlow [8], SciPy [9] and many more, provides the perfect starting point for any data analytics problem. Additionally, to simplify environment setup and package management, it is beneficial to use Anaconda [10], which is a Python and R [11] language distribution with a very powerful built-in package manager specifically aimed for scientific and data analytics applications.

Development in the JupyterLab environment happens in so called Jupyter Notebooks [3]. Notebooks contain the actual source code, documentation, and visualization in separate cells. These cells can be executed together or independently, and they can also be executed programmatically with parameters supplied from outside which is one of the main features we will be leveraging during the implementation of our analytics solution.

[Figure 1] depicts a simple Jupyter Notebook example. The environment has an easy-to-understand graphical interface. From top to bottom, it consists of a header with menus and the most used actions as shortcuts and, below the header, by the notebook`s content in cells. In the first cell it contains some documentation, followed by two cells that contain executable Python code.

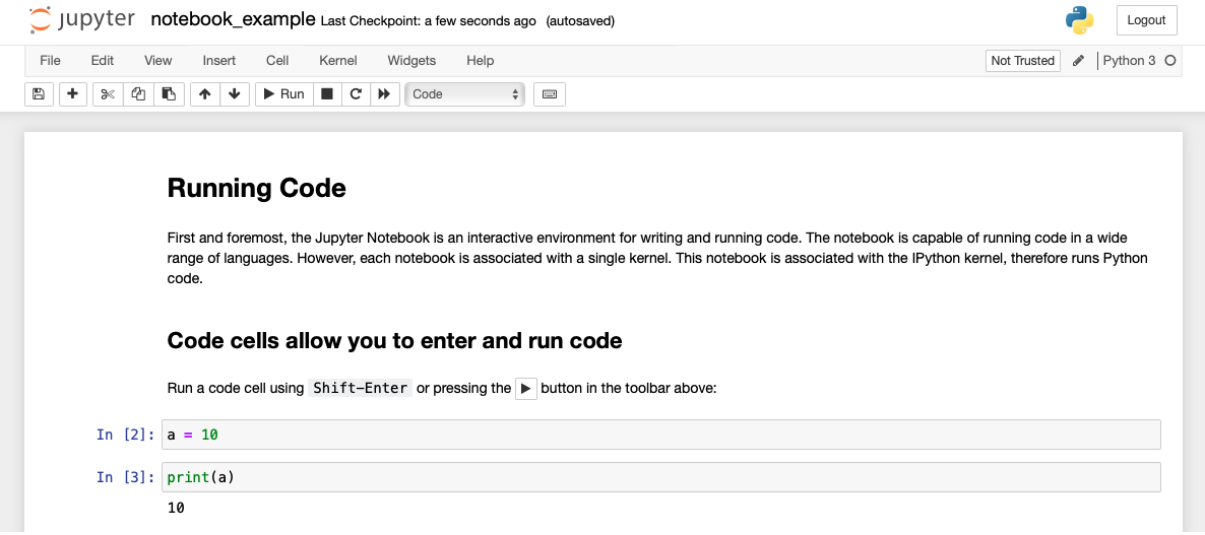

Figure 1 - Jupyter Notebook example

Connecting RTI Connext and JupyterLab

Our main goal is to provide a configurable interface between RTI Connext and the Jupyter environment. Steps to achieve this include examining the requirements, defining an architecture that satisfies the requirements, choosing the right persistence option and finding the right technologies for programming the application that unites everything and provides the interface.

The first and foremost thing to do is to consider what the requirements are for such an application. 1) Modularity, 2) extensibility, 3) rapid response time, and 4) easy configurability are four crucial features.

1) The application must be modular, allowing to change different parts of it without substantial effort, such as changing the way how the application persists data.

2) The application must be extensible, providing an easy way to, for example, subscribe to new DDS topics and start analyzing the received data.

3) The application must respond as soon as possible when new data is received from any of the subscribed topics. As DDS systems are mostly real-time systems, this means real-time support, so different parts of the application must not block it from handling incoming DDS data.

4) The application must provide comprehensive configuration options, which allow users to comfortably configure every important part and feature of it. This includes configuring persistence, subscribed topics, and notebook parametrization.

The following very important thing to do is to define an architecture that satisfies the requirements, and to do so, one of our best options is to utilize software design patterns. There is not a general pattern that fits every application as the architecture must specifically fit the exact requirements, but in general with the help of design patterns, the application can be broken up into logically separated parts allowing development to be more distributed, additionally, the application becomes much more understandable and maintainable.

Data persistence is also a very important feature we must consider. To meet real-time constraints, it is crucial for data to be accessible to notebooks as soon as possible, without any additional delay. We are going to consider two options: 1) persisting DDS data only with the application and using 2) RTI Database Integration Service [12].

1) Persisting DDS data only with our application is a viable solution. It solely consists of saving the subscribed topics` data locally so that it can be immediately provided to notebooks for analysis. But this solution is not very robust and as real-time analysis usually uses recent data, old data might be discarded, which could be useful for further offline analysis.

2) Using RTI Database Integration Service is a perfect option to overcome the shortcomings of the previous solution. It provides great relational database support, so we can conveniently store all our data without having to compromise and discard any part of it. Furthermore, it provides real-time notifications with new table values inside the payloads, which can be used to send data to our application and keep its functionality the same.

[Figure 2] depicts an example high-level architecture that incorporates the aspects we have discussed so far. The application is divided into three main parts: 1) subscribers, 2) persistence and 3) notebook execution.

1) Subscribers subscribe to the necessary topics the application intends to analyze.

2) Persistence is implemented both globally and locally. Global persistence happens with the RTI Database Integration Service, and it provides real-time updates to the application. Locally, the application persists only the data that is necessary for real-time analysis.

3) The application manages notebook execution with commands and provides parameters and data to notebooks. Parameters include such things as source and target file locations. Additionally, besides locally persisted data, notebooks can also receive data from the database, but in real-time scenarios it is advisable to avoid unnecessary delays that stem from database queries.

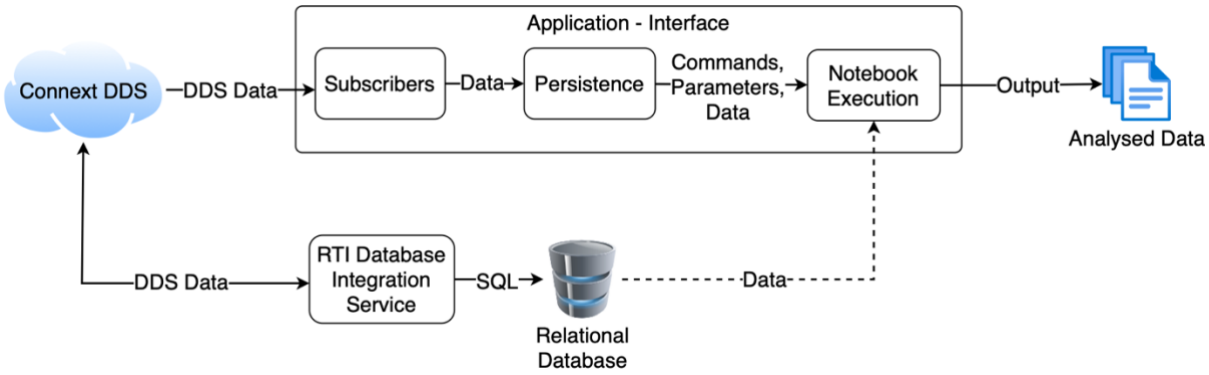

Figure 2 - Example high-level architecture

Finally, we have to choose the right technologies to unite everything we have considered so far. We are going to use Python for programming the application that is going to persist data from the subscribed topics and provide it to notebooks for analysis. To get access to the Connext databus from Python, we are going to utilize RTI Connector [13]. JupyterLab is going to be used to develop the notebooks that will perform data analytics. To execute notebooks from our Python application, a free and open-source library named “papermill” [14] will be used. Papermill provides an easy-to-use interface to parametrize and execute notebooks from code.

Example application

As an example, we will be looking at how the proposed solution can be utilized in an IoT based solar farm system. [Figure 3] depicts the high-level architecture of the system. It consists of solar panels, energy storages, loads and their respective controllers, a database, and our analysis application

which we will be mainly focusing on in this example. All the components communicate with the help of Connext DDS.

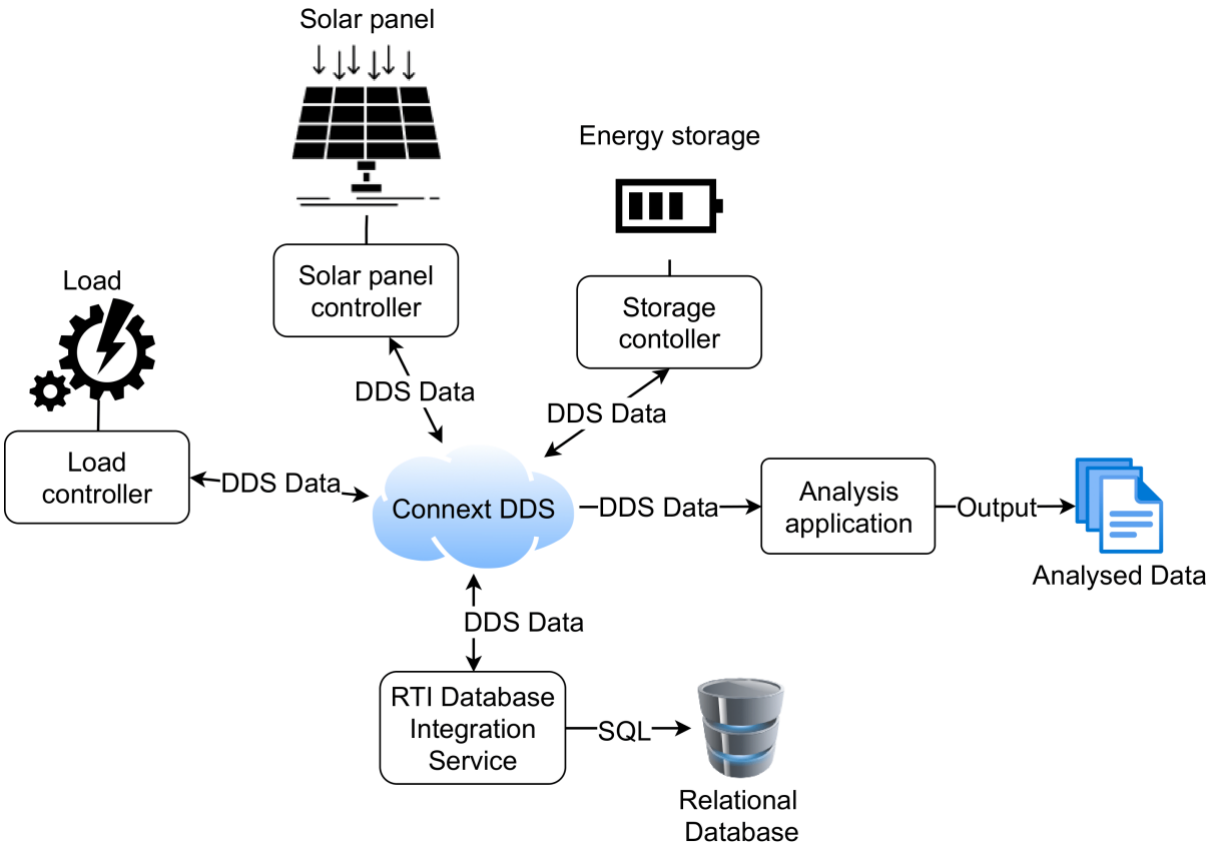

Figure 3 - Solar farm system high-level architecture

Now we are going to define the architecture of the analytics application [Figure 4] that satisfies the requirements we have discussed in the previous topics and go through what role each part of it plays.

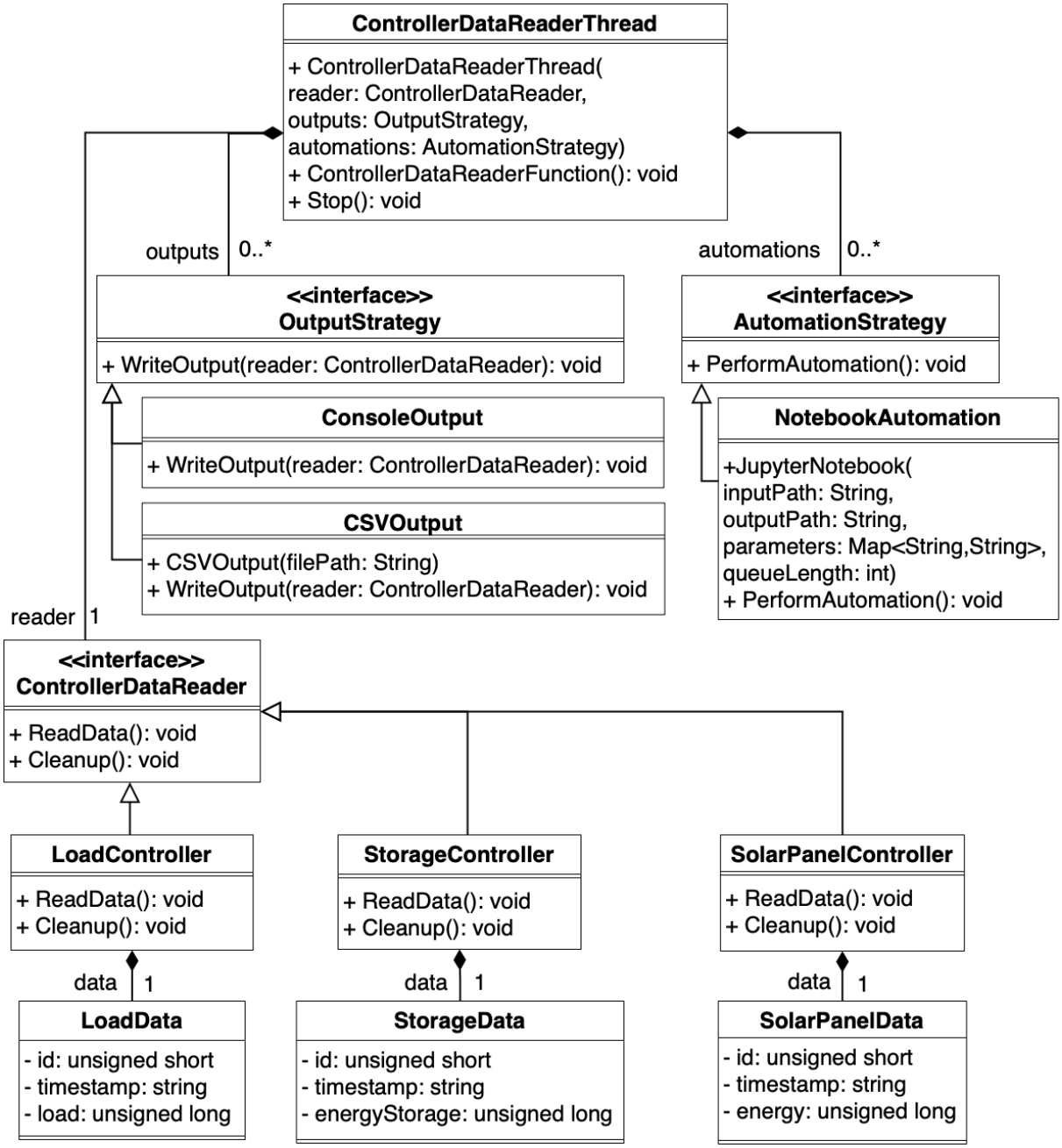

Figure 4 – The architecture of the analytics application

1) ControllerDataReader: Interface that every data reader must implement. With these functions, data readers can be instructed to read from the topics they subscribe to through RTI Connector. There are three data readers respectively to the three controllers [Figure 3].

2) OutputStrategy: Interface that every output handler class must implement. Through this interface, output handlers can be instructed to produce output. There are two output handlers, one writes to the console, the other one into a CSV file.

3) AutomationStrategy: This interface enables the application to perform different automated tasks, in our case it is used to perform data analytics with the help of the “NotebookAutomation” class. It supplies parameters and runs Jupyter Notebooks using the papermill package. Notebook execution frequency can be controlled with the “queueLength” parameter, execution happens only after the specified number of new packets of data have been received.

4) ControllerDataReaderThread: This class holds together everything we have discussed so far. It receives a single reader and multiple output handlers and automated tasks. With the help of this class, every defined reader gets associated to a separate thread, that waits until new data is received from DDS, then executes the given output handlers and automations.

The architecture satisfies the requirements we discussed in the previous sections. With the help of design patterns, the application is highly modular and extensible, new readers can be implemented, and the persistence format of the readers can be conveniently modified by supplying different classes that implement the respective required interfaces. Configuration is convenient as the entire configuration of a reader can take place when a thread class instance is created, additionally, readers do not block each other as they run in different threads.

In the following, we look at how the application performs in action. [Figure 5] depicts a test environment that simulates a small solar farm that consists of three solar panels, a single load, and a single energy storage sending measurements through Connext DDS to the analysis application which analyses the solar panel controllers` data.

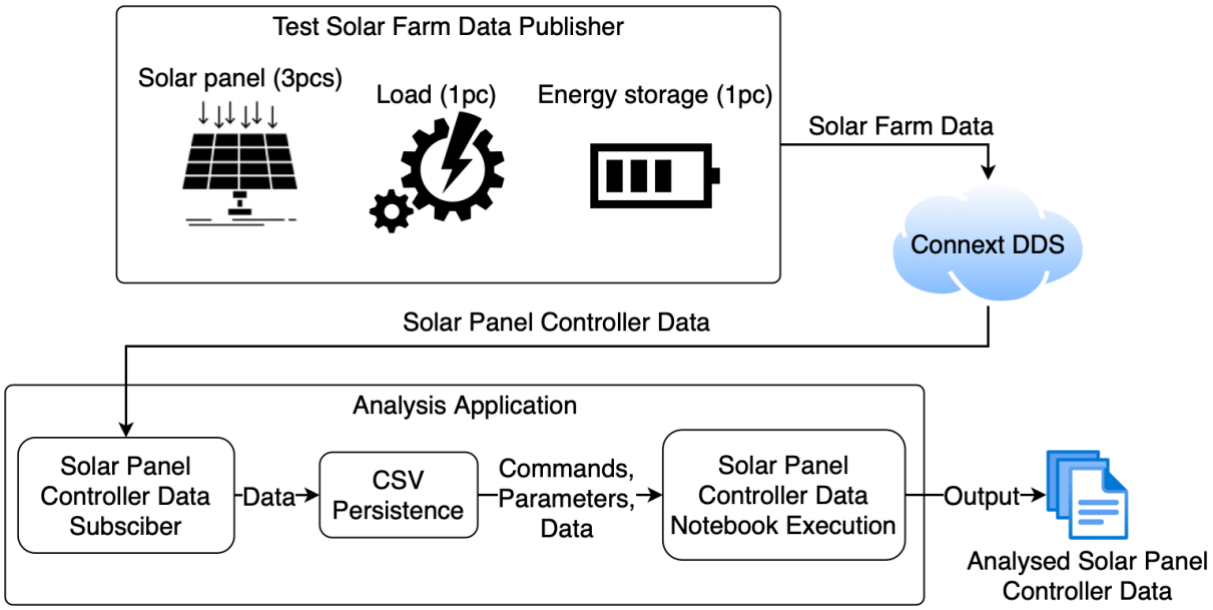

Figure 5 - Example solar farm and application data flow

[Figure 6] depicts the test solar farm data publisher application. Every two seconds it sends mock data about the solar panels, load, and energy storage units of the example solar farm.

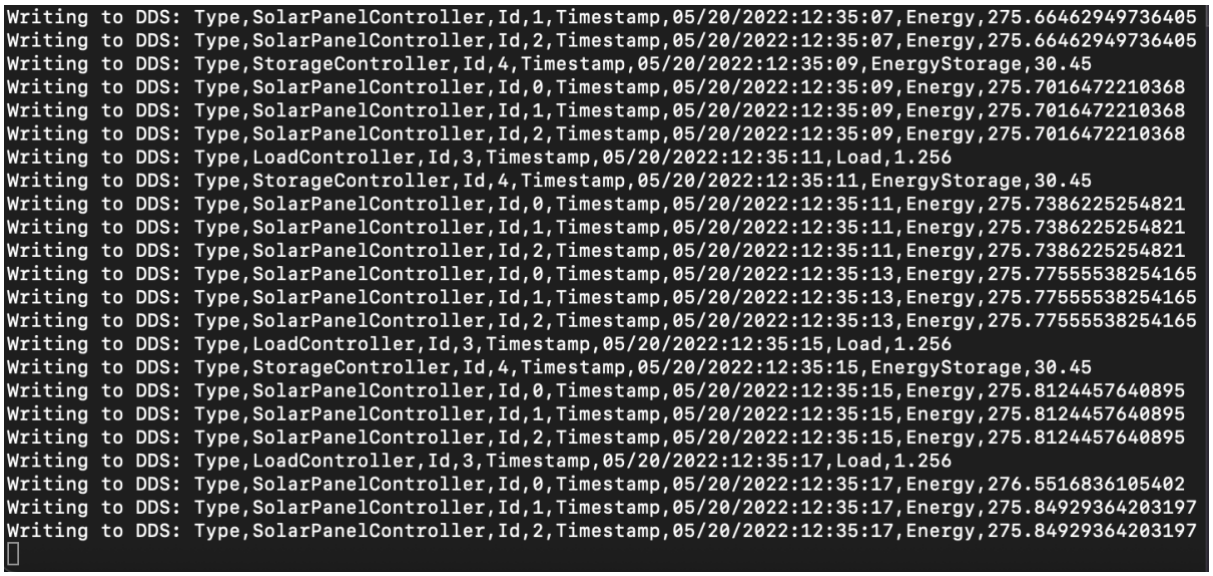

Figure 6 - Providing test data to our analytics application through DDS

The analysis application can be observed in [Figure 7]. It consists of a single reader that is subscribed only to the solar panel controllers` data, moreover it is supplied with a console and a CSV output and a notebook automation that runs whenever new data is received. The CSV output is used to provide data to the supplied notebook that performs data analysis, and its execution is depicted in [Figure 7] in lines starting with “Executing”.

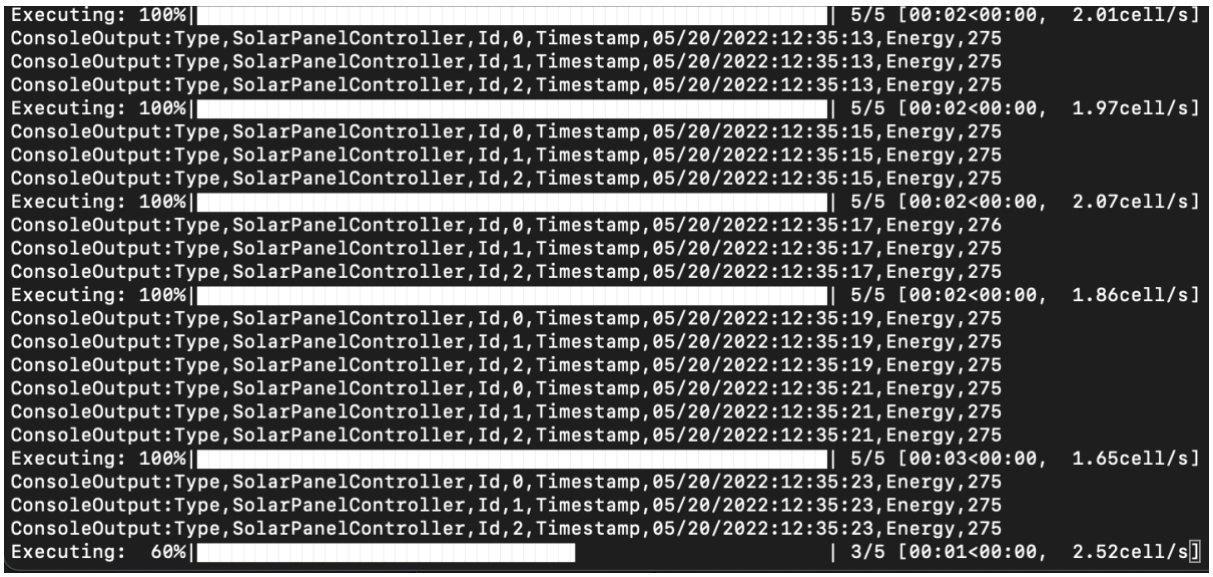

Figure 7 - Reading data from the subscribed topic with our application and executing analysis

Finally, [Figure 8] shows an excerpt from the output. It gets updated every time new solar panel controller data is received, providing real-time insight into the system with the help of Jupyter Notebook.

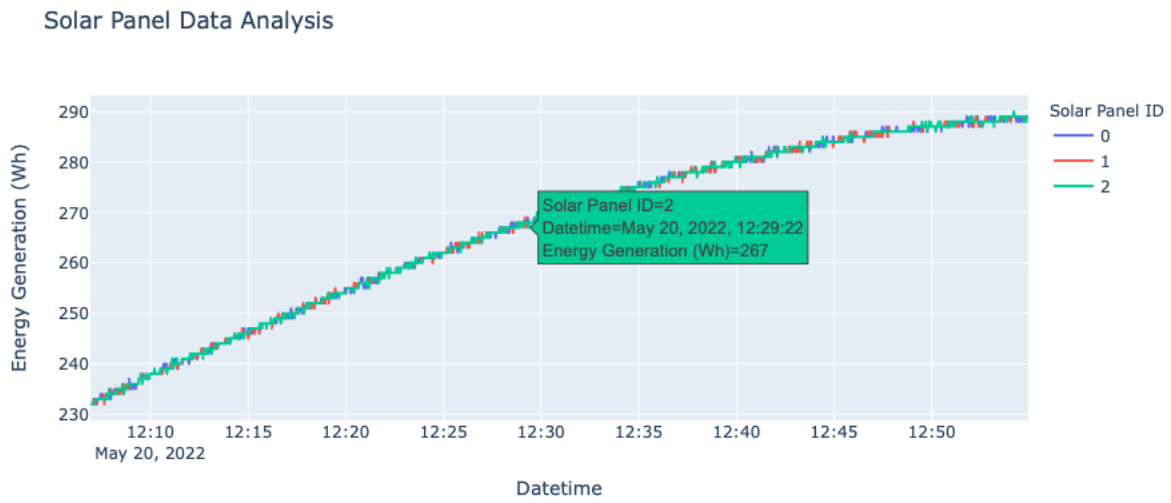

Figure 8 - Excerpt from the notebook after executing analysis

Conclusion

In this article we have taken a brief look at how the power of JupyterLab can be leveraged in DDS systems. Not only is JupyterLab a very robust and flexible tool that provides limitless possibilities when it comes to scientific computing and data analytics, but with tens of thousands of additional packages it significantly eases and speeds up development. We have examined various requirements and aspects of the interface that provides connection between Connext DDS and Jupyter Notebooks, focusing mainly on such features as modularity, flexibility, rapid response time and configurability. Finally, with the help of an example, we looked at how the proposed solution can be leveraged in a real-world system.

Bibliography

[1] Real-Time Innovations Products - Real-Time Innovations, "https://www.rti.com/products," [Online].

[2] Connext DDS Telegraf Plugin - Real-Time Innovations, "https://www.rti.com/developers/rti-labs/telegraf-plugin-for-connext-dds," [Online].

[3] Jupyter Lab - Interactive Computing - Project Jupyter, "https://jupyter.org," [Online].

[4] Python Programming Language - Python Software Foundation, "https://www.python.org," [Online].

[5] Pandas - Data Manipulation and Analysis Software Library - McKinney Wes, "https://pandas.pydata.org," [Online].

[6] NumPy - Python Scientific Computing Library - Travis Oliphant, "https://numpy.org," [Online].

[7] Scikit - Free Machine Learning Library - David Cournapeau, "https://scikit-learn.org/stable/," [Online].

[8] Tensorflow - Free and Open-source Machine Learning and Artificial Intelligence Library - Google Brain Team, "https://www.tensorflow.org," [Online].

[9] SciPy - Free and Open-source Scientific Computing Library - Enthought Inc., "https://scipy.org," [Online].

[10] Anaconda - Data Science Tool - Anaconda Inc., "https://www.anaconda.com," [Online].

[11] R Programming Language - R Core Team, "https://www.r-project.org," [Online].

[12] Real-Time Innovations Database Integration - Real-Time Innovations, "https://www.rti.com/products/add-on-products/database-integration," [Online].

[13] Real-Time Innovations Connector - Real-Time Innovations, "https://www.rti.com/products/tools/connector," [Online].

[14] Papermill - Parametrizing and Executing Jupyter Notebooks - nteract team, "https://papermill.readthedocs.io/en/latest/," [Online].

[15] Data Distribution Service Standard - Object Management Group (OMG), "https://www.dds-foundation.org," [Online].